USER MANUAL

NC Integrator

Integrator User Manual

3xx-603450-xUxx

September 30, 2003

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTEGRATOR GUIDELINES **1**

TURBO PMAC LOOKAHEAD FUNCTION **1**
Introduction 1
Quick Instructions: Setting Up Lookahead 2
Detailed Instructions: Setting up Lookahead 3
Defining the Coordinate System *3*
Lookahead Constraints *3*
Position Limits *3*
Velocity Limits *4*
Acceleration Limits *4*
Calculating the Segmentation Time *4*
Block Rate Relationship *5*
Calculation Implications *5*
Calculating the Required Lookahead Length *6*
Lookahead Length Parameter *7*
Defining the Lookahead Buffer *7*

CUTTER COMPENSATION **9**
Cutter Radius Compensation 9
Defining the Plane of Compensation *9*
Defining the Magnitude of Compensation *9*
Turning on Compensation *10*
Turning off Compensation *10*
How Turbo PMAC Introduces Compensation 10
Inside Corner Introduction *11*
Outside Corner Introduction *12*
Treatment of Inside Corners *13*
Treatment of Outside Corners *13*
Sharp Outside Corner *14*
Shallow Outside Corner *15*
Treatment of Full Reversal *15*
Note on Full Circles *15*
Speed of Compensated Moves *16*
Changes in Compensation *16*
How Turbo PMAC Removes Compensation 18
Inside Corner *18*
Outside Corner *18*
Failures in Cutter Compensation 19
Inability to Calculate Through Corner *19*
Inside Corner Smaller than Radius *20*
Inside Arc Radius Smaller than Cutter Radius *20*

HOW TO MAKE A CUSTOM TOOL OFFSET PAGE **21**
How to Set Parts Counter 22
Parts Total *22*
Parts Required *22*
Parts Count *23*
Integrators Function *23*

HOW TO ADD AND DISPLAY USER MESSAGES **25**
To Set or Reset the Message 25
Clearing Messages *26*

Message Box *26*
Displaying Messages *26*
How to Add/Modify User G, M or T Code 27
Adding G60.1 in MILL.G file *27*
How to Launch a Different Application from NC 28
Launch CALCULATOR from NC *28*

MODIFYING WORK AND TOOL OFFSETS FROM PMAC 29
Triggering PMAC NC to Read or Write an Offset 29
Telling PMAC NC What Offset to Read or Write 29
Where PMAC NC Returns Data from a Read or Write of an Offset 30
Setting a Work Offset *30*
Setting a Tool Offset *31*
Reading a Work Offset *31*
Reading a Tool Offset *32*
Implementation Issues in PLC and Motion Program Code *32*

SAMPLE CONNECTION DIAGRAMS 33

INTEGRATOR GUIDELINES

The Delta Tau NC is an open architecture controller with standard machine software (G, M, T and PLC) that can be modified and with available features that can be added. The instructions discussed in this manual should be followed by an integrator who knows PMAC well.

TURBO PMAC LOOKAHEAD FUNCTION

Introduction

Turbo PMAC can perform highly sophisticated lookahead calculations on programmed trajectories to ensure that the trajectories do not violate specified maximum quantities for the axes involved in the moves. This permits the writing of the motion program simply by describing the commanded path. Vector feedrate becomes a constraint instead of a command; programmed acceleration times are used only to define corner sizes and minimum move block times. Turbo PMAC will control the speed along the path automatically (but without changing the path) to ensure that axis limits are not violated.

Lookahead calculations are appropriate for any execution of a programmed path where throughput has been limited by the need to keep execution slow throughout the path because of the inability to anticipate the few sections where slow execution is required. The lookahead function's ability to anticipate these problem areas permits much faster execution through most of the path, dramatically increasing throughput.

Because of the nature of the lookahead calculations – trajectory calculations are done well in advance of the actual move execution, and moves are kept within machine limits by the automatic adjustment of move speeds and times – they are not appropriate for some applications. Any application requiring quick reaction to external conditions should not use lookahead. In addition, any application requiring precise synchronization to external motion, such as those using PMAC's external time base feature should not use lookahead.

When the lookahead function is enabled, Turbo PMAC will scan ahead in the programmed trajectories, looking for potential violations of its position, velocity, and acceleration limits. If it sees a violation, it will then work backward through the pre-computed buffered trajectories, slowing down the parts of these trajectories necessary to keep the moves within limits. The calculations are completed before these sections of the trajectory are actually executed.

Turbo PMAC can perform these lookahead calculations on **LINEAR** and **CIRCLE** mode moves. The coordinate system must be put in segmentation mode (Isx13 > 0) to enable lookahead calculations, even if only **LINEAR** mode moves are used. (The coordinate system must be in segmentation mode anyway to execute **CIRCLE** mode moves or cutter radius compensation.) In segmentation mode, Turbo PMAC splits the moves into small segments automatically, which are executed as a series of smooth splines to re-create the programmed moves.

Turbo PMAC stores data on these segments in a specially defined lookahead buffer for the coordinate system. Each segment takes Isx13 milliseconds when it is put into the buffer, but this time can be extended if it or some other segment in the buffer violates a velocity or acceleration limit.

This technique permits Turbo PMAC to create deceleration slopes in the middle of programmed moves, at the boundaries of programmed move, or over multiple programmed moves, whichever is required to create the fastest possible move that does not violate constraints. All of this is done automatically and invisibly inside the Turbo PMAC; the programmer and operator do not need to understand the workings of the algorithm.

If Turbo PMAC's inverse kinematic calculations are used, the conversion from tip coordinates to joint coordinates takes place before lookahead calculations, segment by segment for **LINEAR** and **CIRCLE** mode moves. Therefore, Turbo PMAC can execute the lookahead calculations in joint space, motor by motor, even if the system has been programmed in tip coordinates.

Once the lookahead function has been set up, the lookahead function operates transparently to the programmer and the operator. No changes need to be made to a motion program to use the lookahead

function, although the programmer may choose to make some changes to take advantage of the increased performance capabilities that lookahead provides.

Quick Instructions: Setting Up Lookahead

The following list quickly explains the steps required for setting up and using the lookahead function on the Turbo PMAC. Greater detail and context are given in the subsequent section.

1. Assign all desired motors to the coordinate system with axis definition statements.
2. Set Ixx13 and Ixx14 positive and negative position limits, plus Ixx41 desired position-limit band, in counts for each motor in the coordinate system. Set bit 15 of Ixx24 to 1 to enable desired position limits.
3. Set Ixx16 maximum velocity in counts/msec for each motor in the coordinate system.
4. Set Ixx17 maximum acceleration in counts/msec^2 for each motor in the coordinate system.
5. Set Isx13 segmentation time in msec for the coordinate system to minimum programmed move block time or 10 msec, whichever is less.
6. Compute maximum stopping time for each motor as Ixx16/Ixx17.
7. Select motor with longest stopping time.
8. Compute number of segments needed to look ahead as this stopping time divided by (2 * Isx13).
9. Multiply the segments needed by 4/3 (round up if necessary) and set the Isx20 lookahead length parameter to this value.
10. If the application involves high block rates, set the Isx87 default acceleration time to the minimum block time in msec; set the Isx88 default S-curve time to 0.
11. If the application does not involve high block rates, set the Isx87 default acceleration time and the Isx88 default S-curve time parameters to values that give the desired blending corner size and shape at the programmed speeds.
12. Store these parameters to non-volatile memory with the **SAVE** command to make them an automatic part of the machine state.
13. After each power-up/reset, send the card a **DEFINE LOOKAHEAD {# of segments},{# of outputs}** command for the coordinate system, where **{# of segments}** is equal to Isx20 plus any segments for which backup capability is desired, and **{# of outputs}** is at least equal to the number of synchronous M-variable assignments that may need to be buffered over the lookahead length.
14. Load the motion program into the Turbo PMAC. The motion program defines the path to be followed. The lookahead algorithm may reduce the speed along the path, but it will not change the path.
15. Run the motion program, and let the lookahead algorithm do its work.

Detailed Instructions: Setting up Lookahead

A few steps are required to calculate and set up the lookahead function. Typically, the calculations only have to be done once in the initial configuration of the machine. Once configured, the lookahead function operates automatically and invisibly.

Defining the Coordinate System

The lookahead function checks the programmed moves against all motors in the coordinate system. The first step is therefore to define the coordinate system by assigning motors to axes in the coordinate system with axis definition statements. This action is covered in the Setting up the Coordinate System section of the User Guide.

Lookahead Constraints

Turbo PMAC's lookahead algorithm forces the coordinate system to observe four constraints for each motor. These constraints are defined in I-variables for each motor representing maximum position extents, velocities, and accelerations. These I-variables must be set up properly in order for the lookahead algorithm to work properly.

Position Limits

Variables Ixx13 and Ixx14 for each Motor xx define the maximum positive and negative position values, respectively, that are permitted for the motor (software overtravel limits). These variables are defined in counts, and are referenced to the motor zero, or home, position (often called machine zero). Even if the origin of the axis for programming purposes has been offset (often called program zero), the physical position of these position limits does not change; they maintain their reference to the machine zero point. Turbo PMAC checks the actual position for each motor as the trajectory is being executed against these limits; if a limit is exceeded, the program is aborted and the motors are decelerated at the rate set by Ixx15.

Variable Ixx41 for each Motor xx defines the distance between the actual position limits explained above, and the desired position limit that can be checked at move calculation time, even in lookahead. That is, if the calculated desired move position is greater than (Ixx13 – Ixx41), or less than (Ixx14 + Ixx41), this will constitute a desired position limit violation. Desired position limits are only checked if bit 15 of Ixx24 is set to 1.

In this mode, if the lookahead algorithm, while scanning ahead in the programmed trajectory, determines that any motor in the coordinate system would exceed one of its desired position limits, it will suspend the program and force a stop right at that limit. It will then work backwards through the buffered trajectory segments to bring the motors to a stop along the path at that point in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Note

If bit 14 of Ixx24 is also set to 1, the program does not stop at the limit. Instead, it will continue, with the offending motor saturating at the limit value.

When stopped on a desired position limit within lookahead, the program is only suspended, not aborted. The action is effectively equivalent to issuing a \ quick-stop command. It is possible to retrace the path coming into the limit, or even to resume forward execution after changing the limit value. An abort command must be issued before another program can be started.

Note

If an actual position limit is also tripped during the deceleration to a stop at the desired position limit, the program is aborted, so retracing and resuming are not possible. For this reason, if the possibility of retracing and resuming is important, Ixx41 should be set to a large enough value so that the actual position limit is never tripped during a desired position limit stop.

This technique permits these software position limits to be placed just within the hard stops of the machine. Without the desired position limits, the software position limits cannot be detected until the actual trajectory passes the limit. This requires that these limits be placed far enough within the hard stops so that the motors have enough distance to stop after they pass the limits. (When a motor hits a software position limit without lookahead, the deceleration of motors is controlled by Ixx15, not Ixx17, and deceleration is not necessarily along the programmed path.)

Velocity Limits

Variable Ixx16 for each Motor xx defines the magnitude of the maximum velocity permitted for the motor. These variables are defined in the raw PMAC units of counts per millisecond, so a quick conversion must be calculated from the user units (e.g. millimeters per minute).

If the algorithm, while looking ahead in the programmed trajectory, determines that any motor in the coordinate system is being asked to violate its velocity limit, it will slow down the trajectory at that point just enough so that no limit is violated. It will then work backwards through the buffered trajectory segments to create a controlled deceleration along the path to this limited speed in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Note

During the initial move-block calculations, before move data is sent to the lookahead function, a couple of factors can result in commanded velocities lower than what is programmed. First, if the vector feedrate commanded in the motion program with the **F** command exceeds the maximum feedrate parameter Isx85, then Isx85 is used instead. Second, if the move-block time, either specified directly with the TM command, or calculated as vector-distance divided by vector-feedrate, is less than the programmed acceleration time (the larger of TA or 2 * TS), the programmed acceleration time is used instead. This results in a speed less than what was programmed. The lookahead function can further slow these moves, but it cannot speed them up.

Acceleration Limits

Variable Ixx17 for each Motor xx defines the magnitude of the maximum acceleration permitted for the motor. These variables are defined in the raw PMAC units of counts per (millisecond-squared), so a quick conversion must be calculated from the user units (e.g. in/sec^2, or g's).

If the algorithm, while looking ahead in the programmed trajectory, determines that any motor in the coordinate system is being asked to violate its acceleration limit, it will slow down the trajectory at that point just enough so that no limit is violated. It will then work backwards through the buffered trajectory segments to create a controlled deceleration along the path to this limited speed in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Calculating the Segmentation Time

Turbo PMAC's lookahead function operates on intermediate motion segments calculated from the programmed trajectory. An intermediate point for each motor is computed once per segment from the programmed path, and then a fine interpolation using a cubic spline to join these segments is executed at

the servo update rate. The user settable segmentation time is therefore an important parameter for optimization of the lookahead function.

Variable Isx13 for each Coordinate System 'x' defines the time for each intermediate segment in the programmed trajectory, in milliseconds, before it is possibly extended by the lookahead function. Isx13 is an integer value; if a non-integer value is sent, Turbo PMAC will round to the next integer. If Isx13 is set to 0, the coordinate system is not in segmentation mode; no intermediate segments are calculated, and the lookahead function cannot be enabled.

Several issues must be addressed in setting the Isx13 segmentation time. These include its relationship to the maximum block rate, the small interpolation errors it introduces, and its effect on the calculation load of the Turbo PMAC. Each of these is addressed in the following sections.

Block Rate Relationship

In most applications, the Isx13 segmentation time will be set so that it is less than or equal to the minimum block (programmed move) time. Put another way, the segmentation rate defined by Isx13 is usually set greater than or equal to the maximum block rate. For example, if a maximum block rate of 500 blocks per second is desired, the minimum block time is 2 milliseconds, and Isx13 is set to a value no greater than 2.

This relationship holds because blocks of a smaller time than the segmentation time are skipped over as Turbo PMAC looks for the next segment point. While this does not cause any errors, there is no real point in putting these programmed points in the motion program if the controller is going to skip over them. However, some inherit old motion programs with points closer together than is actually required; these users may have reason to set the segmentation time larger than the minimum block time.

Note

The programmed acceleration time sets a limit on the maximum block rate. The move time for a programmed block, even before lookahead, is not permitted to be less than the programmed acceleration time. The programmed acceleration time is the larger of the TA time (TA = Isx87 by default) and twice the TS time (TS = Isx88 by default). In high-block-rate lookahead applications, the TA time is typically set equal to the minimum desired block time, and the TS time is typically set to (because it squares up corners).

Calculation Implications

While smaller Isx13 segmentation times permit higher real maximum block rates and permit more accurate interpolation, they increase the Turbo PMAC computational requirements, particularly when lookahead is active. The following table shows the result of benchmarking tests on the Turbo PMAC and the minimum segmentation times that can be used for a given number of axes executing lookahead calculations.

Number of Axes	Maximum Block Rate (blocks/sec)	Minimum Segmentation Time (msec)
2	2000	1 @ 200%
3	1000	1
4	500	2
5	500	2
6	500	2
8	333	3
12	250	4
16	200	5

Notes:
1. Tests performed on 80 MHz Turbo PMAC.
2. Tests performed at default 2.25 kHz servo update rate.
3. Tests performed with no PMAC motor commutation or current-loop closure.
4. Higher block rates can be done, but segmentation will smooth out features.

Note

Subject to these constraints, the length of the lookahead is subject only to memory limitations in the Turbo PMAC.

In general, the Isx13 segmentation time is set to the largest value that meets user requirements in each of the above three concerns. However, it is seldom set larger than 10 msec.

Calculating the Required Lookahead Length

In order for the coordinate system to reach maximum performance, it must be looking ahead for the time and distance required for each motor to come to a full stop from maximum speed. Because the lookahead buffer stores motion segments, this lookahead length must be expressed in segments.

To calculate this value, first compute the worst-case time required to stop for each motor in the coordinate system. This value can be obtained by dividing the maximum motor velocity by the maximum motor acceleration. In terms of Turbo PMAC parameters:

$$StopTime(\,m\,sec) = \frac{Ixx16}{Ixx17}$$

Now take the motor with the longest stop time, and divide this time by 2 (because the segments will come in at maximum speed, which takes half the time of ramping down to zero speed). Next, convert this value to a number of segments by dividing by the coordinate system segmentation time:

$$LookaheadLength(\,segs\,) = \frac{StopTime(\,m\,sec)}{2 * Isx13(\,m\,sec\,s\,/\,seg\,)} = \frac{Ixx16}{2 * Ixx17 * Isx13}$$

This is the number of segments in the lookahead buffer that must be always properly computed ahead of time. Because the Turbo PMAC does not recalculate fully the lookahead buffer every segment, actually it must look further ahead than this number of required segments

Lookahead Length Parameter

Variable Isx20 for the coordinate system tells the algorithm how many segments ahead in the program to look. This value is a function of the number of segments that must always be correct in the lookahead buffer (SegmentsNeeded). The formula is:

$$Isx20 = \frac{4}{3} * SegmentsNeeded$$

Setting Isx20 to a value larger than needed does not increase the computational load (although it does increase the time of heaviest computational load while the buffer is filling). However, it does require more memory storage, and it does increase the delay in having the program react to any external conditions.

Setting Isx20 to a value smaller than needed does not cause the limits to be violated. However, it may cause Turbo PMAC to limit speeds more severely than the Ixx16 limits require in order to ensure that acceleration limits are not violated. In addition, a saw-tooth velocity profile may be observed.

Note

Preliminary versions of the Turbo PMAC firmware had three additional parameters controlling the dynamics of the lookahead operation: Isx21, Isx22, and Isx23. In the current versions of the firmware, these values are fixed at 3, 6, and 7, respectively, and the variables have been removed. Isx21 now permits direct control of the lookahead state of operation.

Defining the Lookahead Buffer

In order to use the lookahead function in a Turbo PMAC coordinate system, a lookahead buffer must be defined for that coordinate system, reserving memory for the buffer. This is done with the on-line coordinate-system-specific **DEFINE LOOKAHEAD** command. Because lookahead buffers are not retained through a power down or reset, this command must be issued after every power-up or board reset.

There are two values associated with the **DEFINE LOOKAHEAD** command. The first determines the number of motion segments for each motor in the coordinate system that can be stored in the lookahead buffer. At a minimum, this must be set equal to Isx20.

If this value is set greater than Isx20, the lookahead buffer stores historical data. This data can be used to reverse through the already executed trajectory. If reversal is desired, the buffer should be sized to store enough back segments to cover the desired backup distance. There is no penalty for reserving more memory for these synchronous M-variable assignments than is needed, other than the loss of this memory for other uses.

The room reserved for the segment data in the lookahead buffer is dependent on the number of motors assigned to the coordinate system at the time of the **DEFINE LOOKAHEAD** command. If the number of motors assigned to the coordinate system then changes, the organization of the lookahead buffer will be wrong, and the program will abort with a run-time error on the next move after the coordinate system is changed.

If the coordinate system must be changed during an application that uses lookahead, the lookahead buffer must first be deleted, then defined again after the change. The following motion program code shows how this could be done:

```
DWELL 10                                ; Stop lookahead execution
CMD "&1 DELETE LOOKAHEAD"               ; Delete buffer
CMD "&1 #4->100C"                       ; Assign new motor to C. S. 1
CMD "&1 DEFINE LOOKAHEAD 1000,100"      ; Redefine buffer
DWELL 10                                ; Make sure commands execute
```

The second value associated with the **DEFINE LOOKAHEAD** command determines the number of synchronous M-variable assignments (e.g. **M1==1**) for the coordinate system that can be stored in the lookahead buffer. Synchronous M-variable assignments in the motion program delay the actual assignment of the value to the M-variable until the start of actual execution of the next move in the motion program. Therefore, these actions must be held in a buffer pending execution.

This size of the buffer for these assignments must be at least as great as the largest number of assignments expected during the time for lookahead. There is no penalty for reserving more memory for these synchronous M-variable assignments than is needed, other than the loss of this memory for other uses.

Note

The buffer reserved in this manner for synchronous M-variables under lookahead is distinct from the fixed-size buffer used for synchronous M-variables without lookahead.

For example, the command **&1 DEFINE LOOKAHEAD 500,50** creates a lookahead buffer for Coordinate System 1 that can store 500 segments for each motor assigned to the coordinate system at that time, plus 50 synchronous M-variable assignments.

CUTTER COMPENSATION

Cutter Radius Compensation

Turbo PMAC provides the capability for performing cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z axes, which should be physically perpendicular to each other. The compensation offsets the described path of motion perpendicular to the path by a programmed amount automatically, compensating for the size of the tool. This permits the user to program the path along the edge of the tool, letting Turbo PMAC calculate the tool-center path, based on a radius magnitude that can be specified independently of the program.

Cutter radius compensation is valid only in **LINEAR** and **CIRCLE** move modes. The moves must be specified by **F** (feedrate), not **TM** (move time). Turbo PMAC must be in move segmentation mode (Isx13 > 0) to do this compensation (Isx13 > 0 is required for **CIRCLE** mode anyway.)

Note

In **CIRCLE** mode, a move specification without any center specification results in a linear move. This move is executed correctly without cutter radius compensation active, but if the compensation is active, it will not be applied properly in this case. A linear move must be executed in **LINEAR** mode for proper cutter-radius compensation.

Defining the Plane of Compensation

Several parameters must be specified for the compensation. First, the plane in which the compensation is to be performed must be set using the buffered motion-program **NORMAL** command. Any plane in XYZ-space may be specified. This is done by specifying a vector normal to that plane, with I, J, and K-components parallel to the X, Y, and Z-axes, respectively.

For example, **NORMAL K-1**, by describing a vector parallel to the Z-axis in the negative direction, specifies the XY-plane with the normal right/left sense of the compensation (**NORMAL K1** would also use the XY-plane, but invert the right/left sense). This same command also specifies the plane for circular interpolation. **NORMAL K-1** is the default. The compensation plane should not be changed while compensation is active.

Other common settings are **NORMAL J-1**, which specifies the ZX-plane for compensation, and **NORMAL I-1**, which specifies the YZ-plane. These three settings of the normal vector correspond to RS-274 G-codes G17, G18, and G19, respectively. If implementing G-codes in Turbo PMAC program 1000, incorporate them in `PROG 1000`:

```
N17000 NORMAL K-1 RETURN
N18000 NORMAL J-1 RETURN
N19000 NORMAL I-1 RETURN
```

Defining the Magnitude of Compensation

The magnitude of the compensation – the cutter radius – must be set using the buffered motion program command **CCR{data}** (Cutter Compensation Radius). This command can take either a constant argument (e.g. **CCR0.125**) or an expression in parentheses (e.g. **CCR(P10+0.0625)**). The units of the argument are the user units of the X, Y, and Z-axes. In RS-274 style programs, these commands are often incorporated into tool data D-codes using Turbo PMAC motion program 1003.

Negative and zero values for cutter radius are possible. Note that the behavior in changing between a positive and negative magnitude is different from changing the direction of compensation. See the Changes in Compensation section of this manual. In addition, the behavior in changing between a non-zero magnitude and a zero magnitude is different from turning the compensation on and off. See the appropriate sections of this manual.

Turning on Compensation

The compensation is turned on by buffered motion program command **CC1** (offset left) or **CC2** (offset right). These are equivalent to the RS-274 G-Codes **G41** and **G42**, respectively. If implementing G-Code subroutines in Turbo PMAC motion program 1000, simply incorporate them in PROG 1000:

```
N41000 CC1 RETURN
N42000 CC2 RETURN
```

Turning off Compensation

The compensation is turned off by buffered motion program command **CC0**, which is equivalent to the RS-274 G- Code **G40**. If implementing G-Code subroutines in Turbo PMAC motion program 1000, incorporate them in PROG 1000:

```
N40000 CC0 RETURN
```

How Turbo PMAC Introduces Compensation

Turbo PMAC gradually introduces compensation over the next **LINEAR** or **CIRCLE** mode move following the CC1 or CC2 command that turns on compensation. This lead-in move ends at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

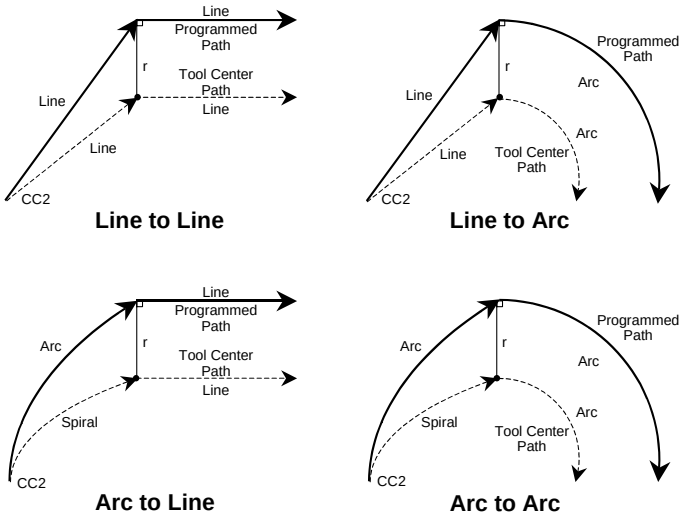
Note

A few controllers can make their lead-in move a CIRCLE-mode move. This capability permits establishing contact with the cutting surface very gently, important for fine finishing cuts.

Inside Corner Introduction

If the lead-in move and the first fully compensated move form an inside corner, the lead-in move goes directly to this point. When the lead-in move is a **`LINEAR`** mode move, the compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a **`CIRCLE`** mode move, the compensated tool path will be a spiral.

Introducing Compensation – Inside Corner

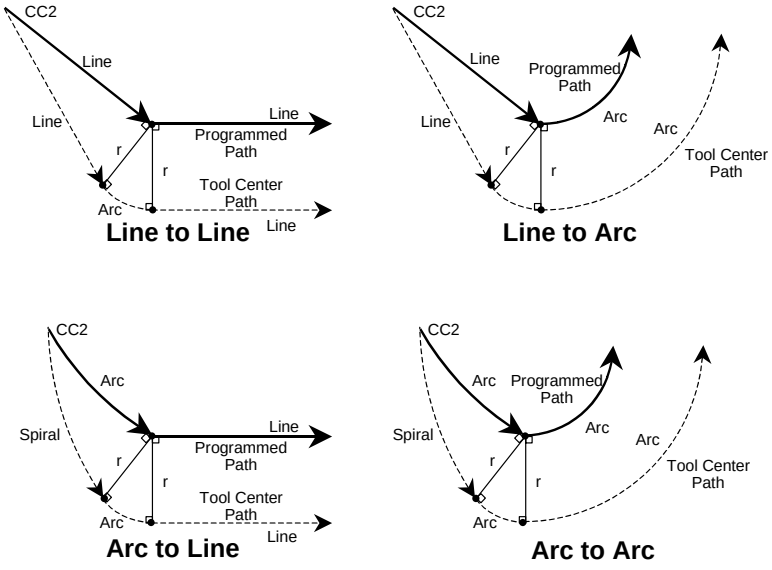


Outside Corner Introduction

If the lead-in move and the first fully compensated move form an outside corner, the lead-in move first moves to a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the lead-in move at the intersection. When the lead-in move is a **`LINEAR`** mode move, this compensated tool path will be at a diagonal to the programmed move path.

When the lead-in move is a **`CIRCLE`** mode move, this compensated tool path will be a spiral. Then a circular arc move with radius equal to the cutter radius is added, ending at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

Introducing Compensation – Outside Corner



Note

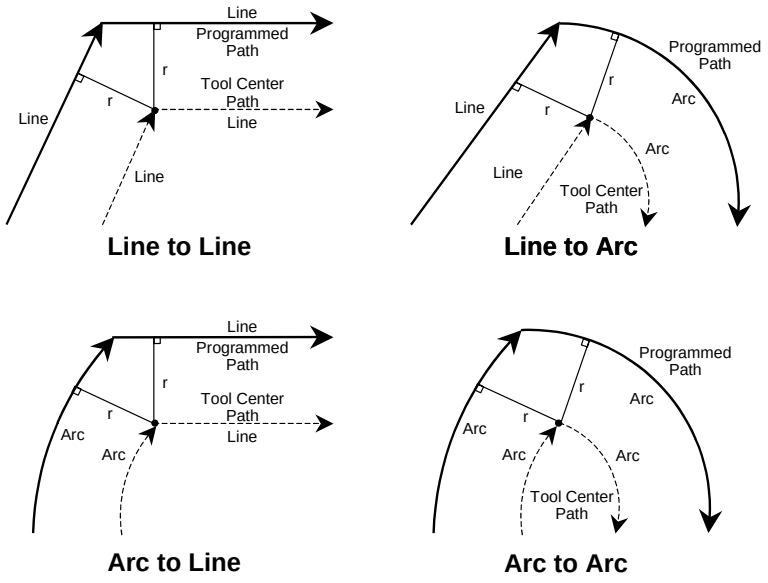
The behavior for lead-in moves is different from changing the compensation radius from zero to a non-zero value while compensation is active. An arc move is always added at the corner, regardless of the setting of Isx99. This ensures that the lead-in move never cuts into the first fully compensated move.

Treatment of Inside Corners

Inside corners are still subject to the blending due to the **TA** and **TS** times in force (default values set by coordinate system I-variables Isx87 and Isx88, respectively). The longer the acceleration time, the larger the rounding of the corner. (The corner rounding starts and ends a distance **F*TA/2** from the compensated, but unblended corner.) The greater the portion of the blending is S-curve, the squarer the corner will be.

When coming to a full stop (e.g. Step, Quit, or **DWELL** at the corner) at an inside corner, Turbo PMAC will stop at the compensated, but unblended, corner point.

Inside Corner Cutter Compensation



Treatment of Outside Corners

For outside corners, Turbo PMAC will either blend the incoming and outgoing moves directly together, or it will add an arc move to cover the additional distance around the corner. Which option it chooses is dependent on the relative angle of the two moves and the value of I-variable Isx99.

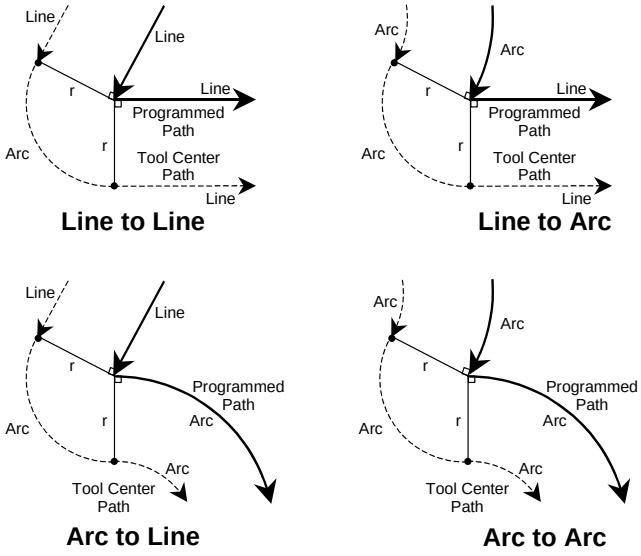
The relative angle between the two moves is expressed as the change in directed angle of the motion vector in the plane of compensation. If the two moves are in exactly the same direction, the change in directed angle is 0^o; if there is a right angle corner, the change is $+/-90^o$; if there is a complete reversal, the change in directed angle is 180^o.

Isx99 specifies the boundary angle between directly blended outside corners and added-arc outside corners. It is expressed as the cosine of the change in the directed angle of motion ($\cos 0^o=1.0$, $\cos 90^o=0.0$, $\cos 180^o=-1.0$) at the boundary of the programmed moves. The change in directed angle is equal to 180^o minus the included angle at the corner.

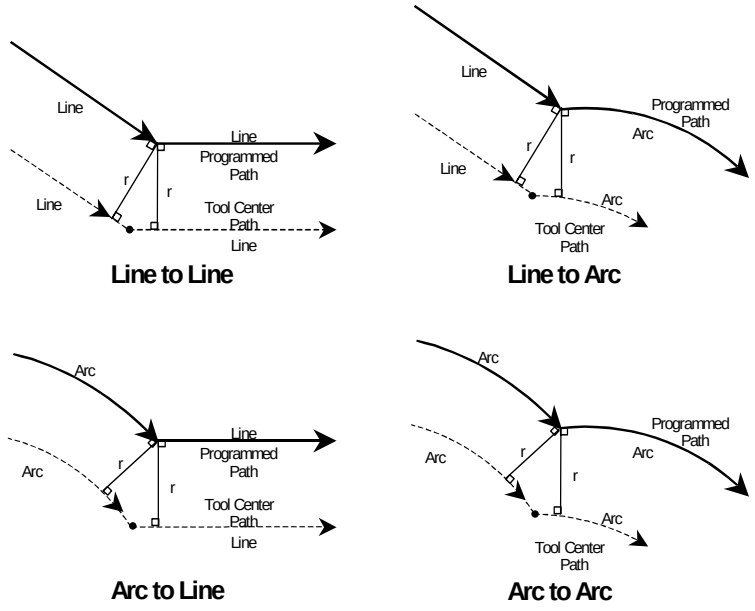
Sharp Outside Corner

If the cosine of the change in directed angle is less than Isx99, which means the corner is sharper than the specified angle, then an arc move will be added around the outside of the corner.

Outside Corner Cutter Compensation, Sharp Angle (cos ΔΘ < Isx99)



Outside Corner Cutter Compensation, Shallow Angle (cos ΔΘ > Isx99)



Shallow Outside Corner

If the cosine of the change in directed angle is greater than Isx99, which means that the corner is flatter than the specified angle, the moves will be directly blended together without an added arc.

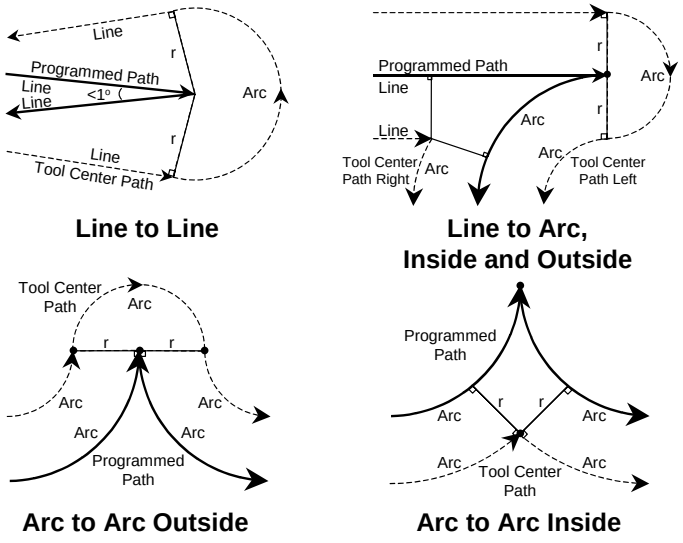
The added arc prevents the compensated corner from extending too far out on the outside of a sharp corner. However, as an added move, it has the minimum time of the acceleration time, which can cause a slowdown on a very shallow angle. While the default value for Isx99 of 0.9998 ($\cos 1^o$) causes an arc to be added on any change in angle greater than 1^o, many will set Isx99 to 0.707 ($\cos 45^o$) or 0.0 ($\cos 90^o$) so arcs are only added on sharp corners.

When coming to a full stop (e.g. Step, Quit, /, or **DWELL**) at an outside corner with an added arc, Turbo PMAC will include the added arc move before stopping. When coming to a full stop at an outside corner without an added arc, Turbo PMAC will stop at the compensated, but unblended, corner point.

Treatment of Full Reversal

If the change in directed angle at the boundary between two successive compensated moves is $180^o \pm 1^o$ (the included angle is less than 1^o), this is considered a full reversal and special rules apply. If both the incoming and outgoing moves are lines, the corner is always considered an outside corner, and an arc move of approximately 180^o is added. If one or both of the moves is an arc, Turbo PMAC will check for possible inside intersection of the compensated moves. If such an intersection is found, the corner will be treated as an inside corner. Otherwise, it will be treated as an outside corner with an added 180^o arc move.

Reversal In Cutter Compensation



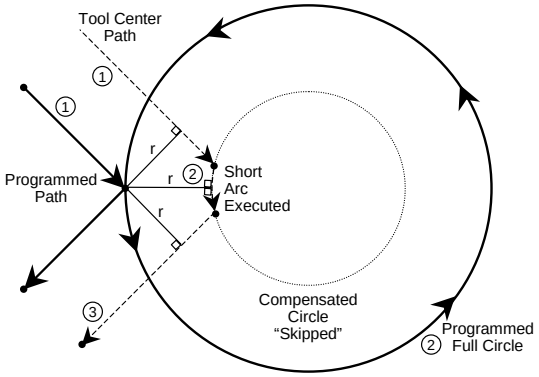
Note on Full Circles

If a full-circle move is executed while in cutter compensation, and one or both of the ends produces a shallow outside corner that is directly blended (no added arc – see the previous section, Treatment of Outside Corners.), the compensated arc move will be extended beyond 360^o, and Turbo PMAC may produce just a very short arc, 360^o shorter than what is desired (making it appear that the circle has been "skipped".

Typically, while this is the result of sloppy programming – an outside corner with a full circle causes an overcut into the circle – many machine designers may want to permit slight cases of this. Coordinate system parameter Isx97 defines the shortest arc angle that may be executed; the longest arc angle is 360^o plus this angle.

The default value of Isx97 sets a minimum arc angle of one-millionth of a semi-circle, enough to account for numerical round off, but sometimes not enough for compensated full circles. To handle these cases, Isx97 should be set to a somewhat larger value.

Failure When Compensation Extends Full Circle



Speed of Compensated Moves

Tool center speed for the compensated path remains the same as that programmed by the F parameter. On an arc move, this means that the tool edge speed (the part of the tool in contact with the part) will be different from that programmed by the fraction R_{tool}/R_{arc}.

Changes in Compensation

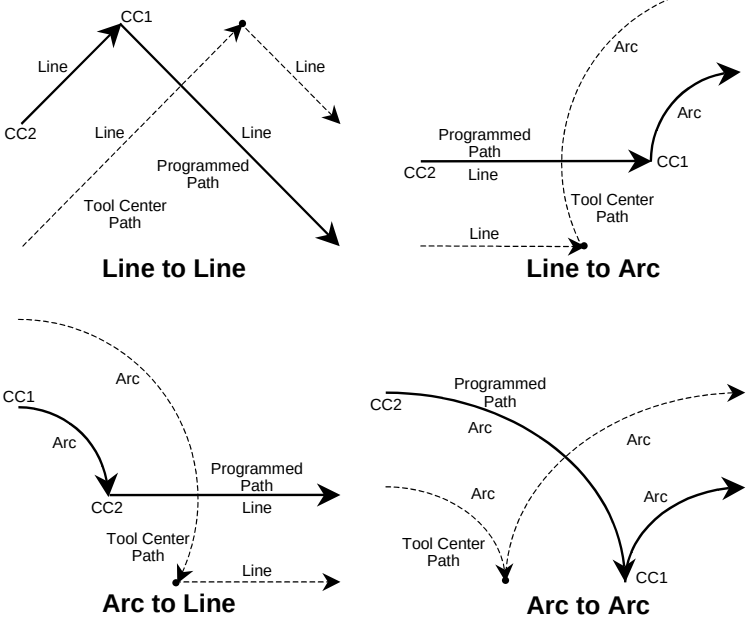
Radius Magnitude Changes

Changes in the magnitude of compensation (new CCR values) made while compensation is active are introduced linearly over the next move. When this change is introduced over the course of a **`LINEAR`** mode move, the compensated tool path will be at a diagonal to the programmed move path. When this change is introduced over the course of a **`CIRCLE`** mode move, the compensated tool path will be a spiral.

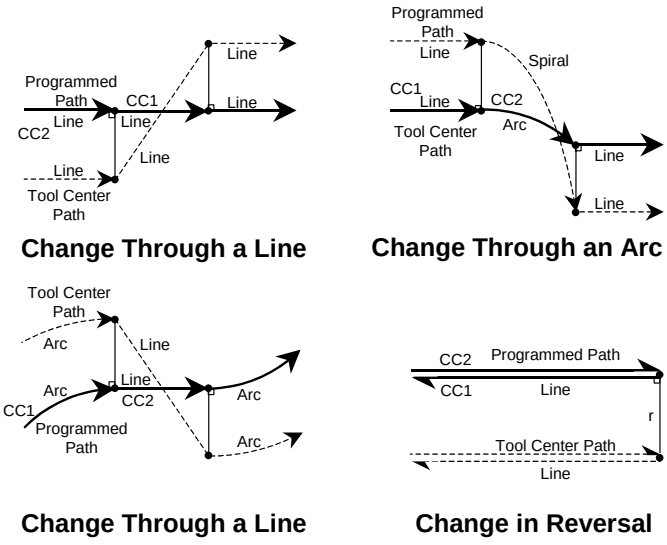
Compensation Direction Changes

Changes in the direction of compensation (between CC1 and CC2) made while compensation are generally introduced at the boundary between the two moves. However, if there is no intersection between the two compensated move paths, the change is introduced linearly over the next move.

Cutter Compensation Change of Direction



Cutter Compensation Change of Direction – No Intersection



How Turbo PMAC Removes Compensation

Turbo PMAC gradually removes compensation over the next **LINEAR** or **CIRCLE** mode move following the **CC0** command that turns off compensation. This lead-out move starts at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

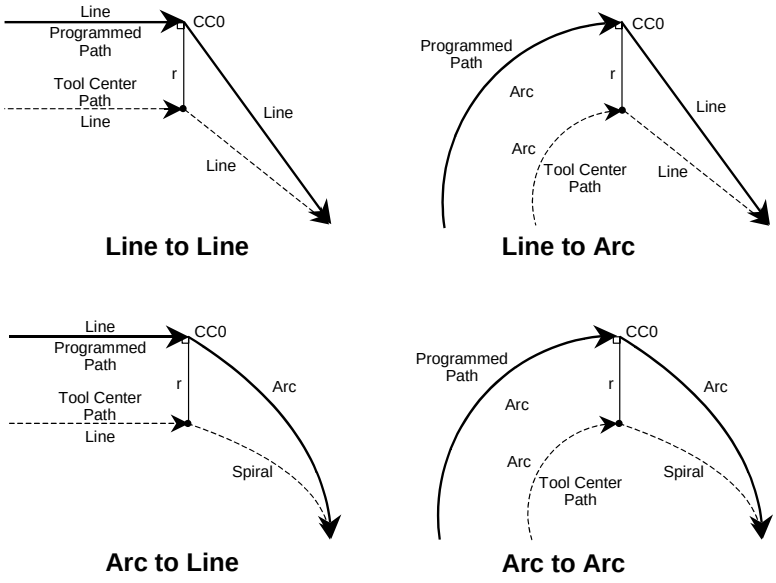
Note

A few controllers can make their lead-out move a **CIRCLE** mode move. This capability permits releasing contact with the cutting surface very gently, important for fine finishing cuts.

Inside Corner

If the last fully compensated move and the lead-out move form an inside corner, the lead-out move starts directly from this point to the programmed endpoint. When the lead-out move is a **LINEAR** mode move, the compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a **CIRCLE** mode move, the compensated tool path will be a spiral.

Removing Compensation – Inside Corner



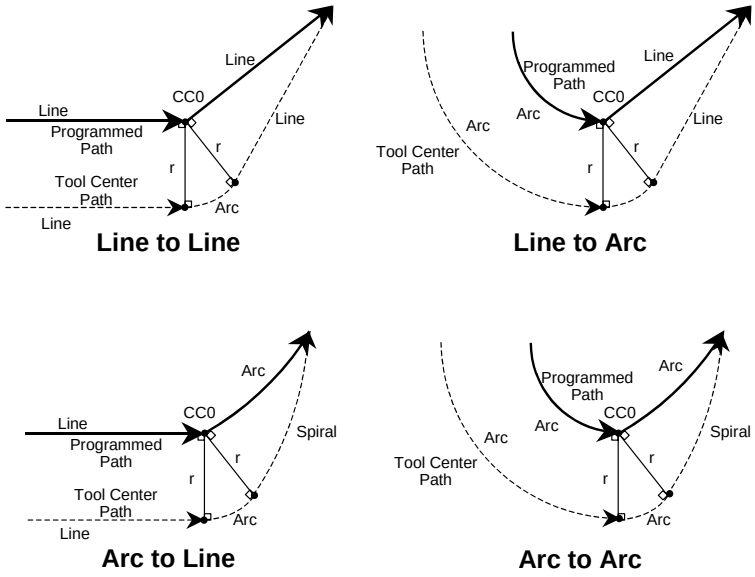
Outside Corner

If the last fully compensated move and the lead-out move form an outside corner, the last fully compensated move ends at a point one cutter radius away from the intersection of the last fully compensated move and the lead-out move, with the line from the programmed point to this compensated point being perpendicular to the path of the fully compensated move at the intersection.

Turbo PMAC then adds a circular arc move with radius equal to the cutter radius, ending at a point one cutter radius away from the same, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the lead-out move at the intersection.

Finally, Turbo PMAC gradually removes compensation over the lead-out move itself, ending at the programmed endpoint of the lead-out move. . When the lead-out move is a **LINEAR** mode move, this compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a **CIRCLE** mode move, this compensated tool path will be a spiral.

Removing Compensation – Outside Corner



Note

This behavior is different from changing the magnitude of the compensation radius to zero while leaving compensation active. An arc move is always added at the corner, regardless of the setting of Isx99. This ensures that the lead-out move will never cut into the last fully compensated move.

Failures in Cutter Compensation

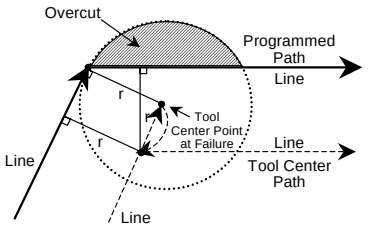
It is possible to give Turbo PMAC a program sequence in which the cutter compensation algorithm will fail not producing desired results. There are three reasons the compensation can fail:

Inability to Calculate Through Corner

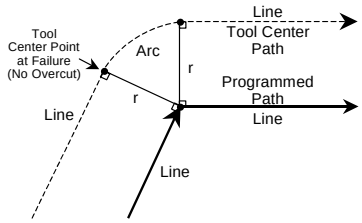
First, if Turbo PMAC cannot see ahead far enough in the program to find the next move with a component in the plane of compensation before the present move is calculated, then it will not be able to compute the intersection point between the two moves. This can happen for several reasons:

- There is a move with no component in the plane of compensation (i.e., perpendicular to the plane of compensation, as in a Z-axis-only move during XY compensation) before the next move in the plane of compensation, and no CCBUFFER compensation block buffer declared.
- There are more moves with no component in the plane of compensation before the next move in the plane of compensation than the CCBUFFER compensation block buffer can hold.
- There are more than 10 **DWELL**s before the next move in the plane of compensation.
- Program logic causes a break in blending moves (e.g. looping twice through a **WHILE** loop).

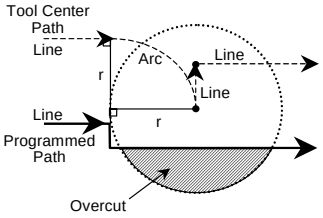
Failures in Cutter Compensation



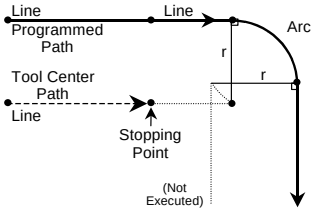
Failure to See Through Inside Corner



Failure to See Through Outside Corner



Inside Corner Smaller Than Cutter Radius



Arc Radius Smaller Than Cutter Radius

If Turbo PMAC cannot find the next move in time, it will end the current move as if the intersection with the next move would form an outside corner. If the next move, when found, does create an outside corner, or continues straight on, compensation will be correct. On an outside corner, an arc move is always added at the corner, regardless of the setting of Isx99. However, if the next move creates an inside corner, the path will have overcut into the corner. In this case, Turbo PMAC will then move to the correct intersection position and continue with the next move, leaving the overcutting localized to the corner.

Inside Corner Smaller than Radius

Second, if the compensated path produces an inside corner with one of the moves shorter than the cutter radius, the cutter compensation will not work properly. This situation results in a compensated move that is in the opposite direction from that of the uncompensated move, and there will be overcutting at the corner.

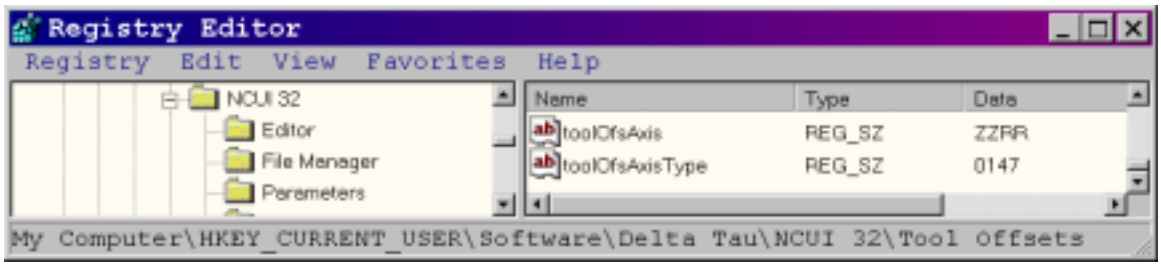
Inside Arc Radius Smaller than Cutter Radius

Third, if the program requests an arc move with compensation to the inside, and the programmed arc radius is smaller than the cutter radius, then no proper path can be calculated. In this case, Turbo PMAC ends the program at the end of the previous move with a run-time error, setting the internal run-time error code in register Y:$002x14 to 7.

HOW TO MAKE A CUSTOM TOOL OFFSET PAGE

The default tool page comes with Z Geom, Z Wear, CC Wear, CC Geom etc.

More columns can be added for other axes in this tool page by changing registry settings. Use the registry editor to go to the Tool Offset Key. (See Screen 1.)



Screen 1 — Tool Offset Key

There are two keys: **toolOffset** and **toolOfsAxisType**. In the figure, Z represents the axis name and R represents the radius. The number of characters/digits must be same in these two keys. Currently there are four characters/digit in these keys.

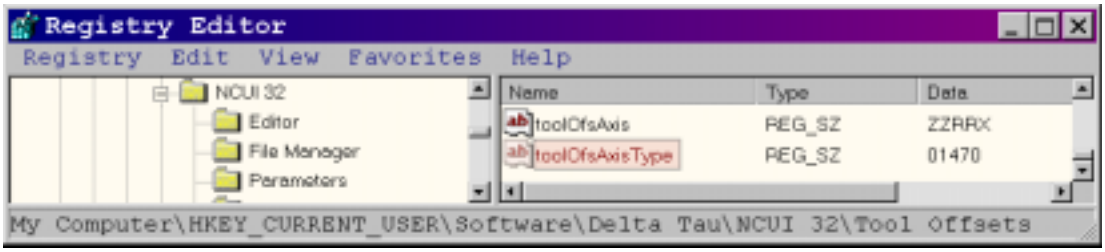
Up to seven types can be assigned to the axis. The table below shows the settings, possible ToolOfsAxis letters, and associated function.

ToolOfsAxisType	Function	ToolOfsAxis
0	Geometry	X,Y Z
1	Wear	X,Y,Z
2	Length	L
3	Orientation	X,Y,Z
4	Cutter Compensation	R
5	Pocket	P
7	Cutter Comp wear	R

To add Geom for X axis, change the registry to:

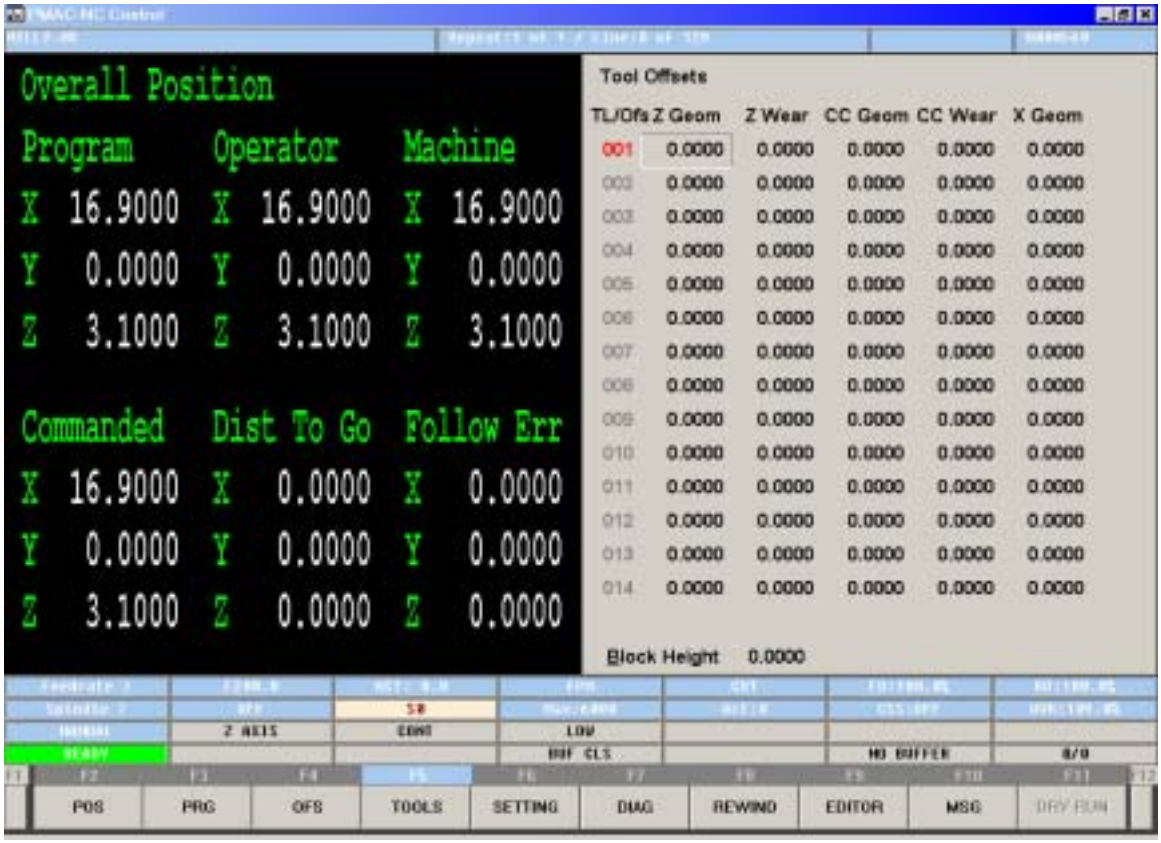
toolOfsAxis = ZZRRX
toolOfsAxisType = 01470

The modified registry will look like:



Screen 2 — Setting Tool Offset

The modified tool page will look like:



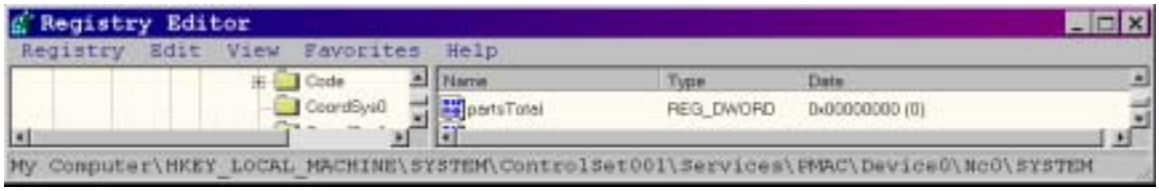
Screen 3 — Modified Tool Page

How to Set Parts Counter

Parts Total

This value is incremented by 1 when a M02, M30 or M code specified by the machine tool builder is executed. Usually this value represents how many parts have been made by the machine since its last rebuild. This value cannot be set on the screen, but is set through the following registry entry:

"partsTotal" = dword:00000000



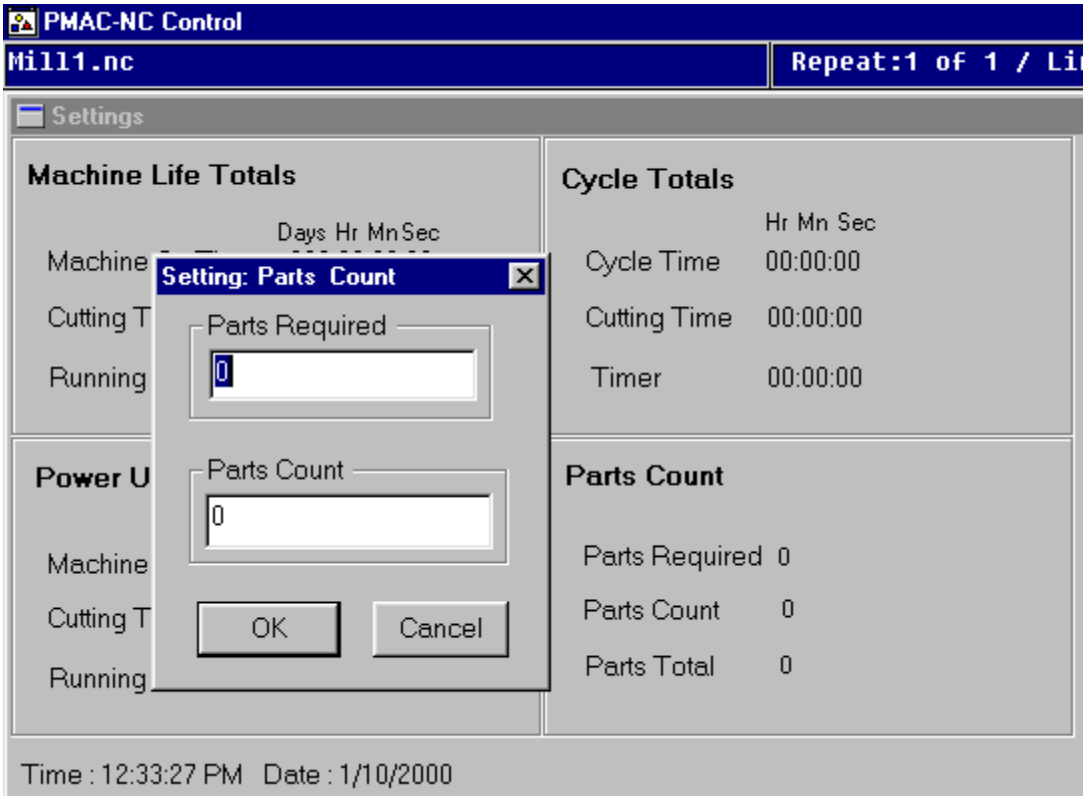
Screen 4 — Parts Total

Parts Required

This value is used for setting the number of machined parts required. When the parts count reaches this value, a machine tool builder specific function may occur.

Parts Count

This value is incremented by 1 when a M02,M30 or a M code specified by the machine tool builder is executed. In general, this value may be reset when it reaches the number of parts required. However, a machine tool builder program specific function may occur.



Screen 5 — Parts Count Setting

To modify the parts required or parts count, press F1 for the settings page, then select F3 Parts Ctr**.**

Integrators Function

PMAC NC DPRAM command words **CS_COMMAND6_M** and **CS_STATUS6_M** contain bits that allow the machine builder to program specific functionality for the parts counter. The bit **CS_PARTS_COUNT** in **CS_COMMAND6_M** allows the parts count and parts total field on the parts counter screen to increment by 1. The bit **CS_PARTS_RESET** in **CS_COMMAND6_M** allows the parts count to be reset to 0. In addition, the bit **CS_PARTS_RESET in CS_STATUS6_M** indicates the parts count has reached the parts required.

Example: M2/M30 code inside the file MILL.M

```
 N2000
N30000
// Cause parts counter PLC to do its work
PARTS_COUNT = 1
RETURN
```

Example: Parts counter PLC

```
OPEN PLC 12 CLEAR
 IF(PARTS_COUNT = 1)
// Cause Parts count and parts total to increment
   @SET_ON(CS_COMMAND6_M,CS_PARTS_COUNT)
  // Wait for increment command to complete
   WHILE(@ON(CS_COMMAND6_M,CS_PARTS_COUNT))ENDW
// Check to see if the parts count has reached the
                     // parts required
   IF(@ON(CS_STATUS6_M,CS_PARTS_RESET))
    // TO DO :
    // User may want to have special action
    // at this location.  Such as set an output
    // to a BEACON to indicate that the programmed
    // required parts have been reached
    // Reset Parts count
     @SET_ON(CS_STATUS6_M,CS_PARTS_RESET)
   ENDIF
   PARTS_COUNT = 0
 ENDIF
CLOSE
-----------------------------------------------
```

CS_COMMAND6_M - CS_PARTS_COUNT Setting this bit in PLC causes the parts count and parts total to increment by 1. This bit is cleared automatically once PMAC NC-32 has incremented the parts total and parts count.

CS_STATUS6_M - CS_PARTS_COUNT This is set by PMAC NC-32 while incrementing the parts total and parts count. This bit is cleared once PMAC NC-32 has incremented the parts total and parts count.

CS_COMMAND6_M - CS_PARTS_RESET Setting this bit in PLC causes the parts count to reset to 0. Once the parts count has reset to 0 the **CS_PARTS_RESET** bit in **CS_STATUS6_M** is cleared.

CS_STATUS6_M - CS_PARTS_RESET This is set by PMAC NC-32 when the parts count reaches the parts required set previously on the settings page.

HOW TO ADD AND DISPLAY USER MESSAGES

There are four different types of user messages possible – Fatal Error, Error, Warning, and Message box. These messages are displayed in NC with different colors indicating different actions. The user messages are written in the ERRORS.DAT file.

Fatal Errors are displayed as ALARM, and can be edited under the [FATAL] section in ERRORS.DAT file.

Errors are displayed as INHIBIT, and can be edited under the [STOP] section in ERRORS.DAT file.

Warnings are displayed as WARNING, and can be edited under the [WARNING] section in ERRORS.DAT file.

Messages are displayed as MESSAGE, and can be edited under the [MESSAGE] section in ERRORS.DAT file.

To set or reset these messages, set the bit of one of the pre-defined M variable macros. These are found in C:\Program Files\Common Files\Delta Tau Shared\address.h file.

The following list explains the association between the type of the error and macro name to be used in PLC or in G, M, T code. 64 messages are possible in each category. One M variable can display 32 error messages.

Display	**PLC or G, M, or T Code**
Fatal Error	ES_ERR_FATAL_M ES_ERR_FATAL2_M
Error	ES_ERR_STOP_M ES_ERR_STOP2_M
Warning	ES_ERR_WARN_M ES_ERR_WARN2_M
General Message	ES_ERR_MSG_M ES_ERR_MSG2_M
Message Box	ES_PLCMSGBOX_M

To Set or Reset the Message

1. Decide how many messages and types of messages.
2. Edit the ERRORS.DAT file for different messages in the editor (Notepad or any editor).
3. Find out the bit address for the message.
4. Define the bit number as a macro.
5. To set the error, use @SET_ON(Message Address, Bit Number) macro in the PLC.
6. To Reset the error, use @SET_OFF(Message Address, Bit Number) macro in the PLC.

For example: The error message No Air Pressure, Check! is displayed if there is no air pressure.

1. Edit the ERRORS.DAT and add a message under [STOP].

```
PLCStopErr64=0064\tPLC\t No Air Pressure, Check!
```

2. Define the bit number as a macro in the header file.

```
#define ERROR_AIR          $80000000          (Bit 32)
```

3. In the PLC, check for air pressure Input and set the error message.

   PLC will look like:

```
IF  (INPUT_AIR = 0)
  @SET_ON(ES_ERR_STOP2_M,ERROR_AIR)
ELSE
  @SET_OFF(ES_ERR_STOP2_M,ERROR_AIR)
ENDIF
```

Clearing Messages

To clear just this message, type:

```
@SET_OFF(ES_ERR_STOP2_M,ERROR_AIR)
```

To clear all messages at once, type:

```
ES_ERR_STOP2_M = $0
```

Use this as an example and display other messages.

Message Box

With the Delta Tau NC, POP UP Windows can be added using message box type messages through a PLC. The PLC can then take action based on the response.

There are two types of message boxes available:

- Message Box with OK button. Messages ranging from 1 - 64 are reserved for this type.
- Message Box with YES and NO buttons. Messages ranging from 65 - 128 are reserved for this type.

A message box is displayed from a PLC by setting variable ES_PLCMSGBOX_M. The messages are stored in ERRORS.DAT (this file) under the [MessageBox] section.

- The message box with an OK button will return 0.
- The message box with a YES button will return 1. (Check 65536 - MSB word Bit 1 in PLC)
- The message box with a NO button will return –1. (check -65536 - Bit 1 MSB word Bit 1 in PLC)

Displaying Messages

To display messages:

1. Set ES_PLCMSGBOX_M = 1 in PLC for MsgBox 0.
2. Set ES_PLCMSGBOX_M = 65 in PLC for MsgBox 64.

To display the message box:

1. Type either Yes or No when prompted at the Do you want to switch OFF the POWER? prompt.
2. Edit the ERRORS.DAT and add a message under the [MessageBox] section:

   PLCMessageBoxErr0= Do you want to switch OFF the POWER.
3. Define the bit number as macro in the header file.

   ```
   #define Msg_box1        65
   ```

The PLC will look like:

```
IF  (Switch_Press  = 1)
  @SET_ON(ES_PLCMSGBOX_M , Msg_box1)
ENDIF
```

```
IF (ES_PLCMSGBOX_M = 65536)    // User responds YES.
   POWER_OFF
ENDIF
IF (ES_PLCMSGBOX_M = -65536)    // User responds NO.
   CONTINUE
ENDIF
```

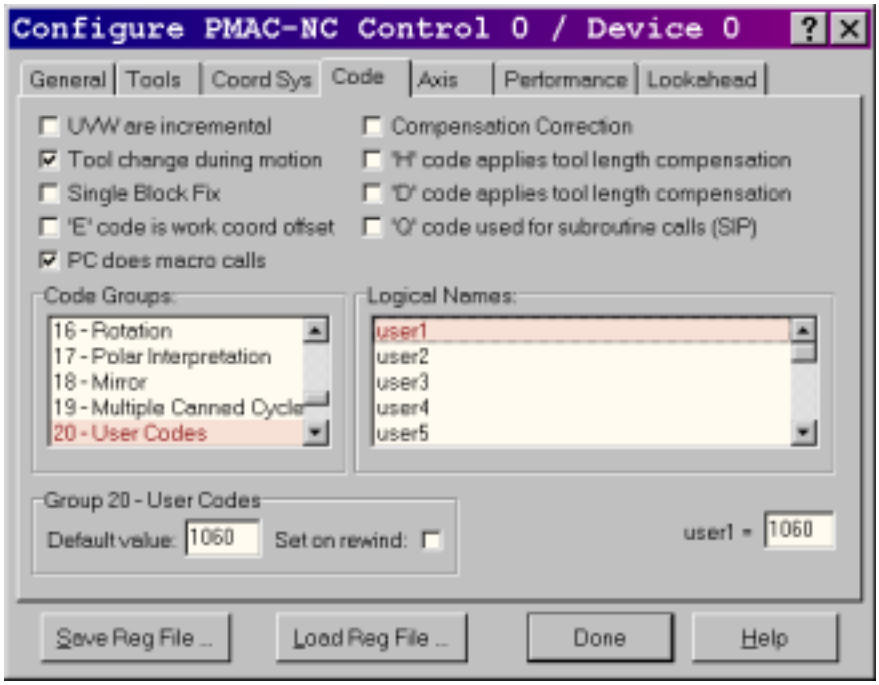
How to Add/Modify User G, M or T Code

User G codes and existing T codes can be modified in the MILL.G or MILL.T files.

The NC software will download these G codes, but will not display them on the screen under Active G codes.

The user G codes can be written from G60.1 to G79.1.

Make sure the Motion application/ NCRegistry application has following settings.



Screen 6 — Code Tab

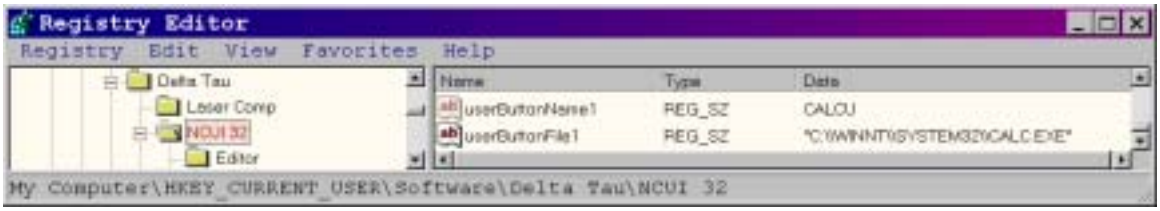
Adding G60.1 in MILL.G file

To add G60.1 to the MILL.G file:

1. Open the MILL.G file in the editor.
2. Add label N60100 and write the code under this label.
3. Follow the same procedure for other user G codes.

How to Launch a Different Application from NC

Delta Tau NC allows launching different applications. To accomplish this, the integrator must add a registry entry in the window registry using the windows registry editor.



Screen 7 — Registry Editor

Launch CALCULATOR from NC

In the registry editor, create userButtonName1 which is a type string. Edit this key and add CALCU**,** which is the name of the user button displayed in NC. This button can be found under PROG – F12. The name of the button should be no more than six characters.

In the registry editor, create userButtonFile1 (another type string). Edit this key and add the executable file path as C:\\WINNT\\SYSTEM32\\CALC.EXE.

To add another userButton**,** increment the number from 1 to 2. Up to seven applications can be launched.

MODIFYING WORK AND TOOL OFFSETS FROM PMAC

PMAC NC contains bits in DPRAM as well as words that allow changing the tool and work offset database within PMAC NC. In addition, a floating point location resides there that is used to exchange floating point data.

Triggering PMAC NC to Read or Write an Offset

The DPRAM location in PMAC $DDFD ($60DFD for Turbo and 0x37F4 for a PC Offset) contains bits that PMAC NC monitors for triggering the modification of an offset. In PMAC the macro variable name PR_BITS_M*(M160)* is assigned to this location. Table 1 indicates how bits in PR_BITS_M trigger various commands. Once the command has been executed, Bit 0 will be cleared by PMAC NC.

PR_BITS_M/Function	Bit 32 – Bit 3	Bit 2	Bit 1	Bit 0
Read An Offset	Don't Care	0	0	1
Write An Offset	Don't Care	0	1	1
No Command pending in PC	Don't Care	D.C.	D.C.	0
Command is pending in PC	Don't Care	D.C.	D.C.	1
Beep PC Speaker	Don't Care	1	0	0

Table 1

Telling PMAC NC What Offset to Read or Write

The DPRAM location in PMAC $DDFE ($60DFE for Turbo and 0x37F8 for a PC Offset) contains a DWORD that PMAC NC interprets when it receives a command for triggering the modification of an offset. In PMAC, the macro variable name PR_COMMAND_M*(M161)* is assigned at this location. PR_COMMAND_M is a DWORD, however it can be interpreted as two 16-bit WORDS concatenated together to form the DWORD. The upper WORD reflects what type of offset can be set. Table 2 indicates how the upper 16 bits of PR_COMMAND_M is interpreted when it receives a trigger command from PR_BITS_M. The lower 16 Bits of PR_COMMAND indicate for what tool number to interpret the command. In tool offsets, the PR_COMMAND_M upper word is 40-48 or 50-58, and in cutter compensation, the PR_COMMAND_M upper word is 2 or 3. For work offsets there is a special interpretation of the lower 16 bits. In G54, the value for the low 16 bits should have a base 20 plus the axis number, where the axis number can be 1,2,3,4,5,6 representing X,Y,Z,A,B,C respectively.

PR_COMMAND_M	Value for Bit 31- 16	Value for Bit 15 – 0
Cutter Comp Geometry Offset	2	Tool Number
Cutter Comp Wear Offset	3	Tool Number
Work Offset G54-G59	8	See Table 3
Work Offset G54.1 P1-P48	10	See Table 3
Five Axis Tool Length	11	Tool Number
Rotation G68	12	See Table 4
A Axis Tool Geometry Offset	40	Tool Number
B Axis Tool Geometry Offset	41	Tool Number
C Axis Tool Geometry Offset	42	Tool Number
X Axis Tool Geometry Offset	43	Tool Number
Y Axis Tool Geometry Offset	44	Tool Number
Z Axis Tool Geometry Offset	45	Tool Number
U Axis Tool Geometry Offset	46	Tool Number
V Axis Tool Geometry Offset	47	Tool Number
W Axis Tool Geometry Offset	48	Tool Number
A Axis Tool Wear Offset	50	Tool Number
B Axis Tool Wear Offset	51	Tool Number
C Axis Tool Wear Offset	52	Tool Number

X Axis Tool Wear Offset	53	Tool Number
Y Axis Tool Wear Offset	54	Tool Number
Z Axis Tool Wear Offset	55	Tool Number
U Axis Tool Wear Offset	56	Tool Number
V Axis Tool Wear Offset	57	Tool Number
W Axis Tool Wear Offset	58	Tool Number

Table 2

For example: To modify the G54 X offset, the low 16 bits should be 21. To modify G54 Y, the low 16 bits should be 22. To modify the G54 Z value, the low 16 bits should be 23. For G55 the base is 40, G56 the base is 60, G57 the base is 80, G58 the base is 100 and for G59 then base is 120. For the extended work offsets G54.1 Pn, where n can range from 1-48, the base value is (n-1)*20. See table 3, below.

Offset Type	Base	Value for Bits 15-0 to Set X Axis	Value for Bits 15-0 to Set Y Axis	Value for Bits 15-0 to Set Z Axis	Value for Bits 15-0 to Set A Axis	Value for Bits 15-0 to Set B Axis	Value for Bits 15-0 to Set C Axis
G54	20	21	22	23	24	25	26
G55	40	41	42	43	44	45	46
G56	60	61	63	63	64	65	66
G57	80	81	82	83	84	85	86
G58	100	101	102	103	104	105	106
G59	120	121	122	123	124	125	126
G54.1 P1	0	1	2	3	4	5	6
G54.1 P2	20	21	22	23	24	25	26
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
G54.1 P48	940	941	942	943	944	945	946

Table 3

Offset Type	X Center of Rotation	Y Center of Rotation	Z Center Of Rotation	XY Rotation Angle	YZ Rotation Angle	ZX Rotation Angle
G68	1	2	3	4	5	6

Table 4

Where PMAC NC Returns Data from a Read or Write of an Offset

The DPRAM location in PMAC $DDFF ($60DFF for Turbo and 0x37FC for a PC Offset) contains a floating point value that PMAC NC uses for setting the modification of an offset or feeding data to the PMAC from PMAC NC. In PMAC, the macro variable name PR_DATA_M*(M162)* is assigned at this location that is of the type DPRAM floating point format.

Setting a Work Offset

1. Type the following to set a work offset: **`PR_DATA_M = 30.8`**
2. Set the upper 16 bits of command to 8 based on table 2 so that PMAC NC sets a work offset in the range G54-G59. Set lower 16 bits to 61 based on table 3 so PMAC sets the G56 X work offset.

```
PR_COMMAND_M = $80000 + 61
PR_BITS_M = PR_BITS_M | 3
```

3. Set bits 0 and 1 to trigger PMAC to read the data placed in the data location

4. Type the following in PEWIN32:

```
M162=30.8
M161=$80000+61
M160=M160|3
```

5. To change G54.1 P48 X value to 48.1, type the following:

```
PR_DATA_M = 48.1
PR_COMMAND_M = $A0000 + 941    ; note that A is 10 in hex
PR_BITS_M = PR_BITS_M | 3
```

6. Type the following in PEWIN32:

```
M162=48.1
M161=$A0000+941
M160=M160|3
```

Setting a Tool Offset

1. To set the Z axis tool Geometry for offset number 8 to 10.0, type the following:

```
PR_DATA_M = 10.0
PR_COMMAND_M = $2D0000 + 8  ; note that 2D is 45 in hex
PR_BITS_M = PR_BITS_M | 3
```

2. Type the following in PEWIN32:

```
M162=10.0
M161=$2D0000+8
M160=M160|3
```

Reading a Work Offset

1. Set upper 16 bits of command to 8 based on table 2 so that PMAC NC reads a work offset in the range G54-G59.

2. Set the lower 16 bits to 61 based on table 3 so PMAC reads the G56 X work offset.

```
PR_COMMAND_M = $80000 + 61
```

3. Set bit 0 to trigger PMAC NC to write the data we want in the data location

```
PR_BITS_M = PR_BITS_M | 1
```

PR_DATA_M should now contain the G56 X axis work offset.

4. Type the following in PEWIN32:

```
M161=$80000+61
M160=M160|1
M162
```

<Will respond with the G56 X axis work offset>

Reading a Tool Offset

1. To read a tool offset, type the following:

```
PR_COMMAND_M = $2D0000 + 8
```

2. Set bits 0 and 1 to trigger PMAC to read the data placed in data location.

```
PR_BITS_M = PR_BITS_M | 1
```

PR_DATA_M should now contain the Z axis tool geometry offset.

3. Type the following in PEWIN32:

```
M161=$2D0000+8
M160=M160|1
M162
```

<PMAC will respond with the Z axis tool geometry offset for tool number 9>

Implementation Issues in PLC and Motion Program Code

When performing these functions in PLC or a motion program code, the programmer should check to make sure that PR_BITS_M bit 0 is cleared before relying on the data in PR_DATA_M to insure that PMAC NC has successfully completed the transaction.

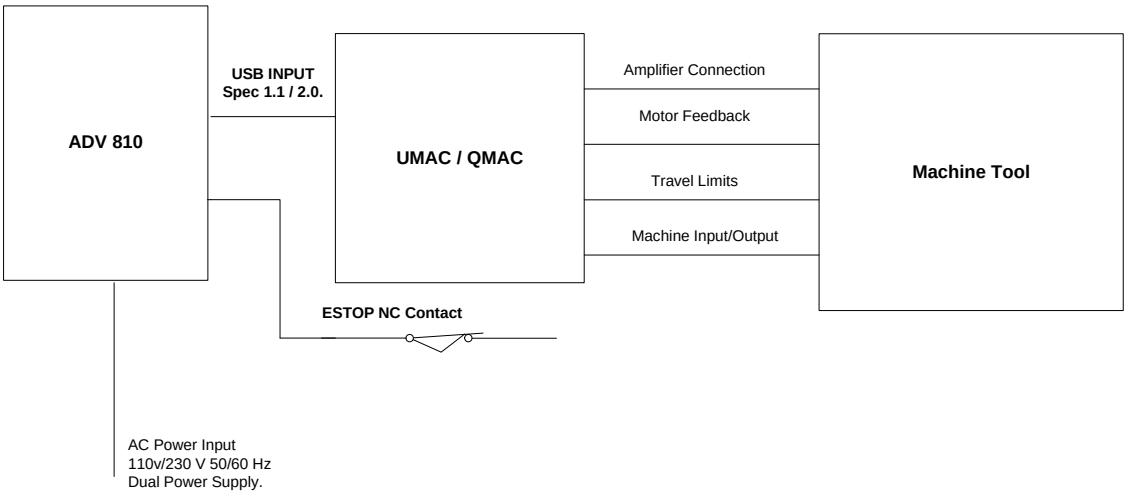
For example: To set tool offset 8 and 9 consecutively, enter the following:

```
PR_DATA_M = 10.0
PR_COMMAND_M = $2D0000 + 8  PR_BITS_M = PR_BITS_M | 3
WHILE (PR_BITS_M&1 != 0)    ; wait for PC to finish
ENDWHILE
PR_DATA_M = 11.0
PR_COMMAND_M = $2D0000 + 9
PR_BITS_M = PR_BITS_M | 3
WHILE (PR_BITS_M&1 != 0)    ; wait for PC to finish
ENDWHILE
```

Production software would of course have a timeout in the while statement and create an error message if the 0^{th} Bit of PR_BITS_M was not cleared.

SAMPLE CONNECTION DIAGRAMS

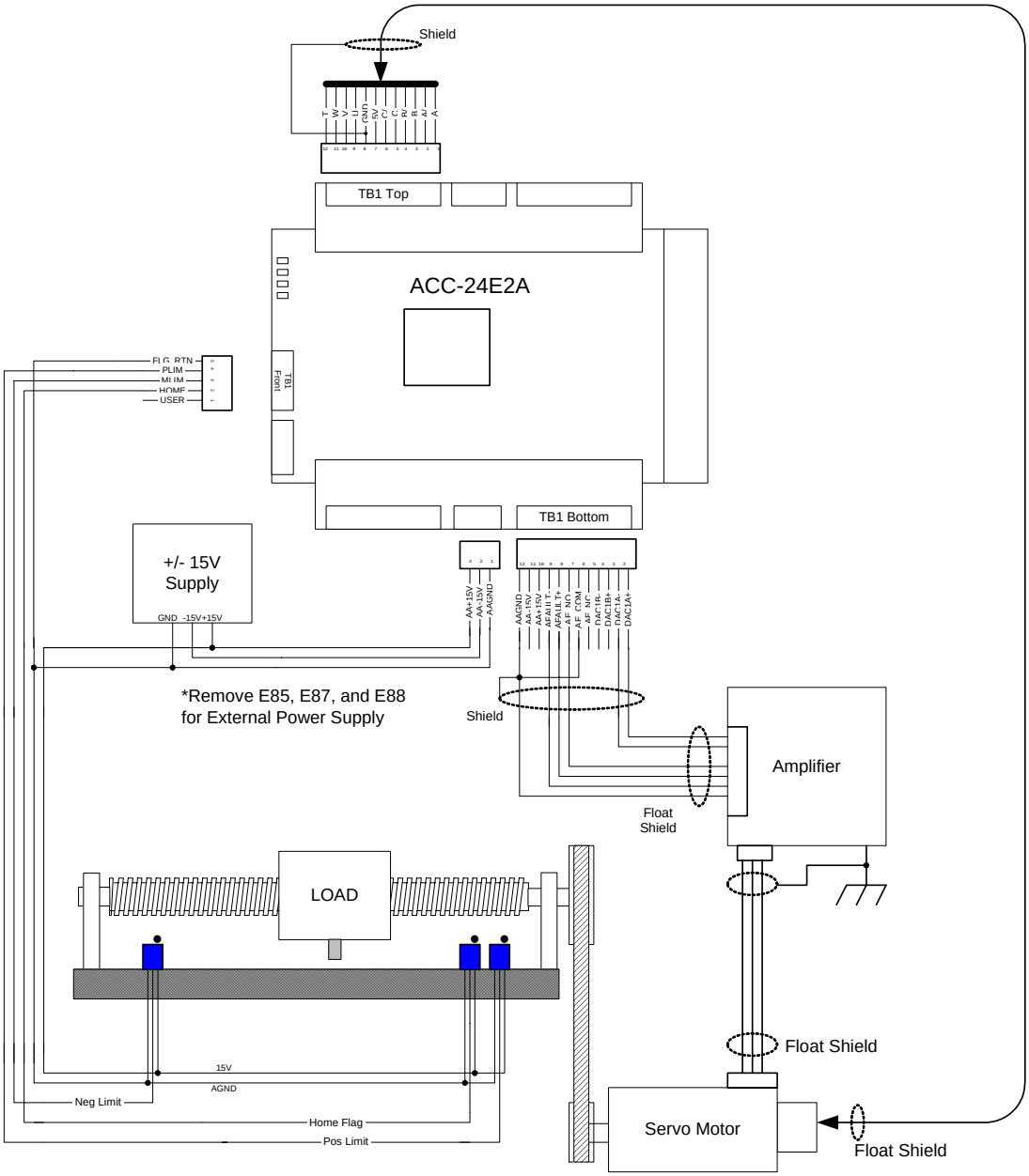
Basic Connection



Adv 810 shipped with three cables - ESTOP with NC (Normally connect), AC Power Input and USB cable.

Diagram 1

NC System Interface - Sample Block Diagram using ACC24E2A

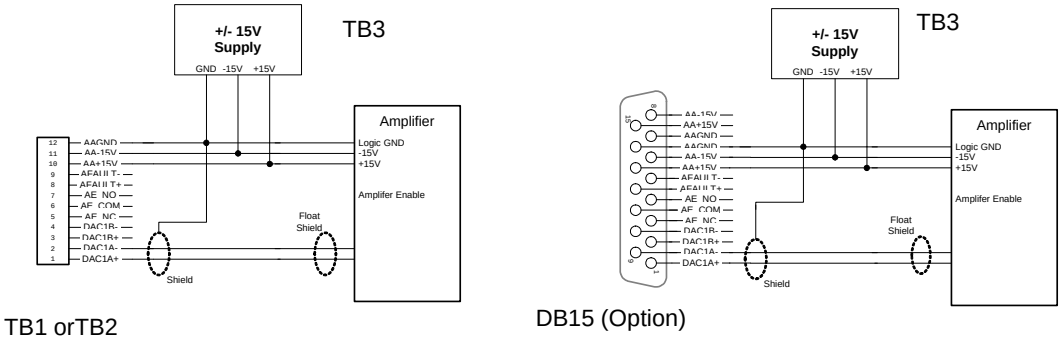


This is a general example of a system with sourcing flags and normally open amplifier enable output from the ACC-24E2A. For opto-isolation an external power supply is used and E85, E87, and E88 have been removed from the ACC-24E2A.

Diagram 2

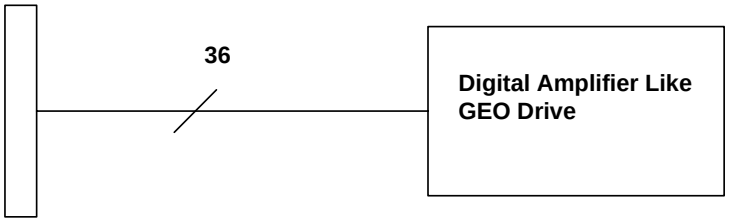
Sample Diagram for Connecting UMAC - Analog AMPLIFIER - ADV810 Connections

ACC-24E2A DAC-Torque/Velocity Mode



Sample Diagram for Connecting UMAC/QMAC - Digital AMPLIFIER - ADV810 Connections

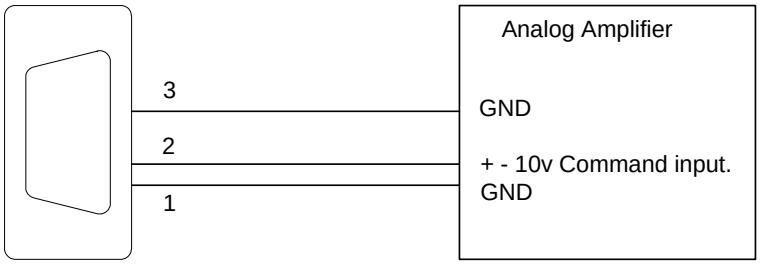
ACC-24E Direct Digital PWM Mode



36 Pin Mini D Connector

This is a standard 36-pin cable compatible with most of the Direct Digital Amplifier manufacturers. Delta Tau Amplifier version is GEO Drive. If GEO drive is not used, check the Pin out in the manual.

Sample Diagram for connecting QMAC - Analog AMPLIFIER - ADV810 Connections



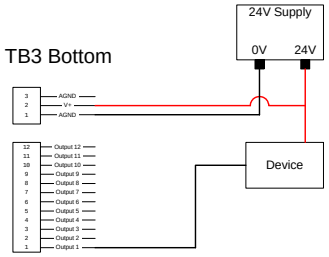
AMP 1 DB9 S - Option A required.

Diagram 3

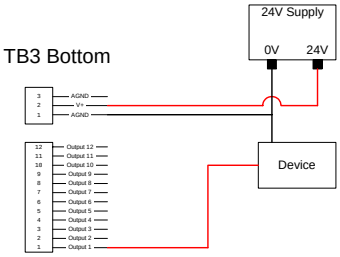
Sample Diagram for Connecting UMAC - INPUT/OUTPUT ACC-65 E

TB1 or TB2 Bottom for Outputs

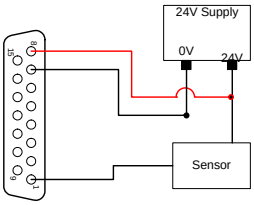
Sinking Outputs



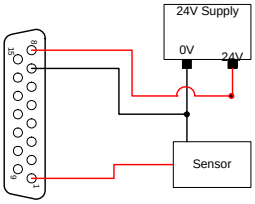
Sourcing Outputs



J1 or J2 Bottom for Outputs

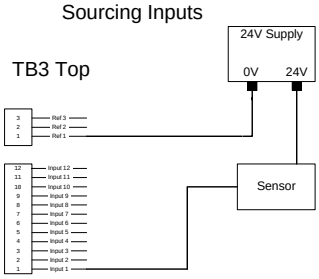


D- Connector as a Option

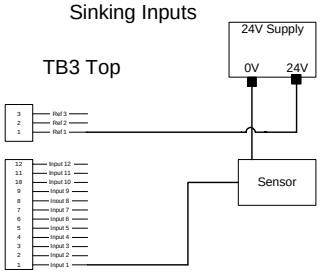


TB1 or TB2 TOP for Inputs

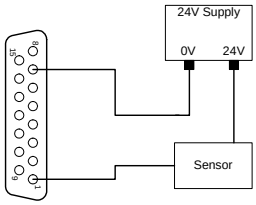
Sourcing Inputs



Sinking Inputs



J1 or J2 TOP for Inputs



D- Connector as a Option

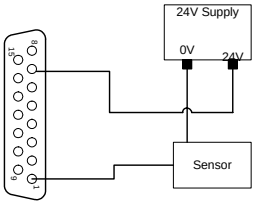
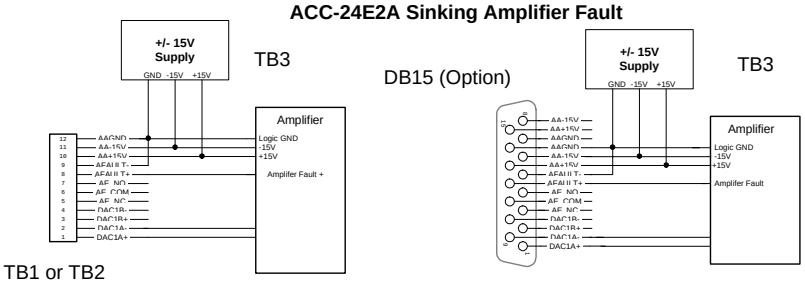


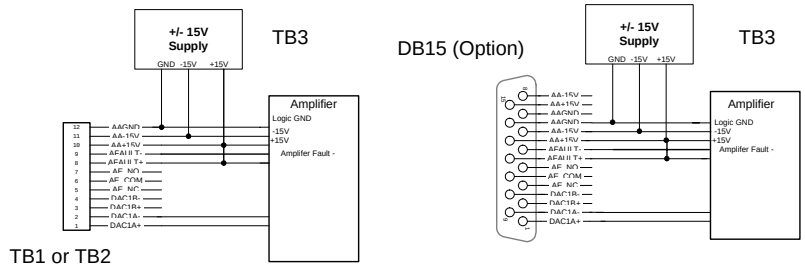
Diagram 4

Sample Diagram for Connecting UMAC - AMPLIFIER FAULT for ADV810

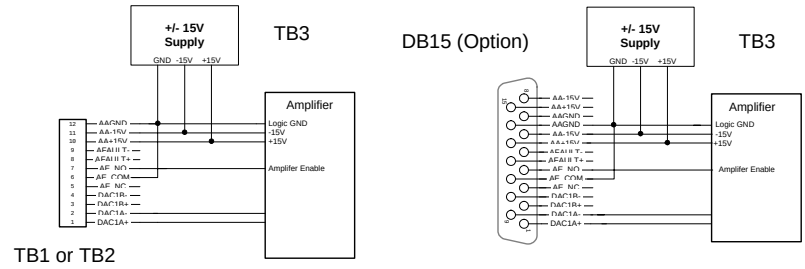
ACC-24E2A Sinking Amplifier Fault



ACC-24E2A Sourcing Amplifier Fault



ACC-24E2A Normally Open Amplifier Enable



ACC-24E2A Normally Closed Amplifier Enable

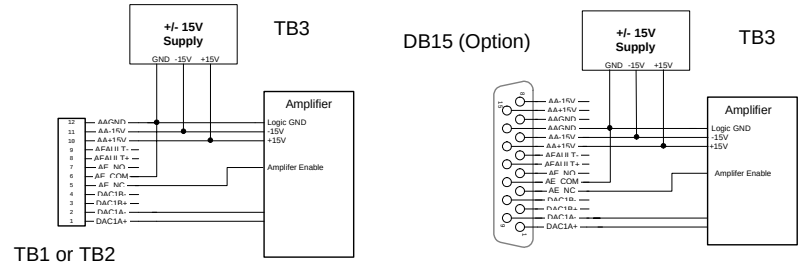
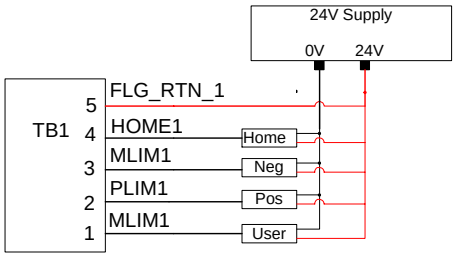


Diagram 5

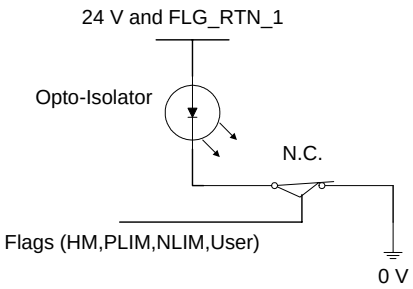
Sample Diagram for Connecting UMAC/QMAC - Machine Flags for ADV810

ACC-24E2A / ACC-24E / QMAC Sinking Flags

Block Diagram

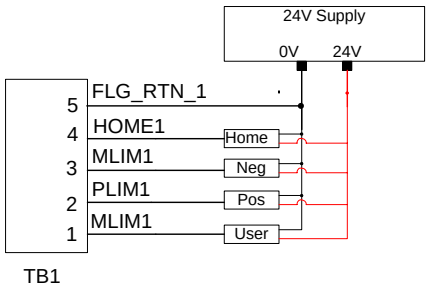


Electrical Equivalent Diagram



ACC-24E2A / ACC-24E / QMAC Sourcing Flags

Block Diagram



Electrical Equivalent Diagram

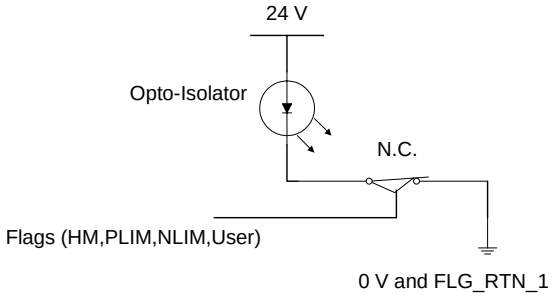
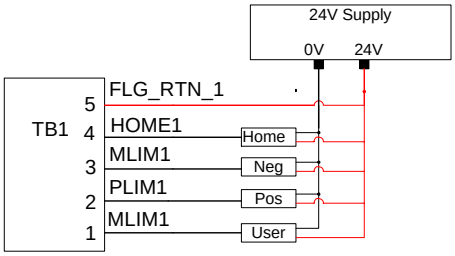


Diagram 6

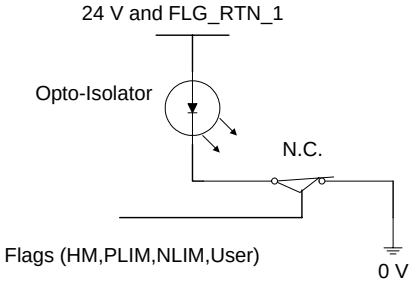
Sample Diagram for Connecting UMAC/QMAC - Machine Flags for ADV810

ACC-24E2A / ACC-24E / QMAC Sinking Flags

Block Diagram

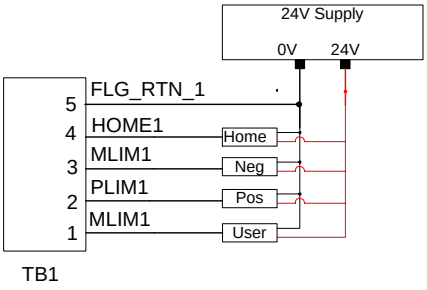


Electrical Equivalent Diagram



ACC-24E2A / ACC-24E / QMAC Sourcing Flags

Block Diagram



Electrical Equivalent Diagram

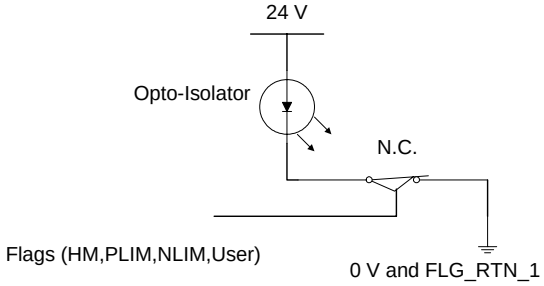
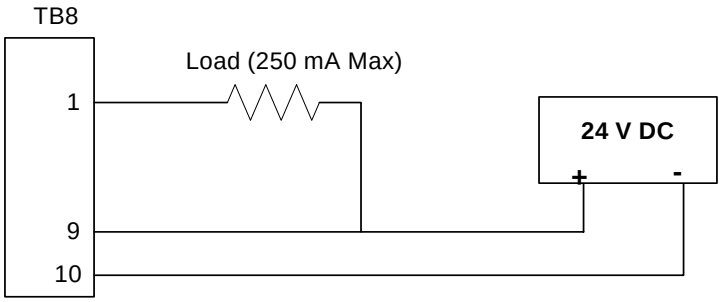


Diagram 7

Sample Diagram for Connecting QMAC - OUTPUT (Sink Only)



Sample Diagram for Connecting QMAC - INPUT TB6 or TB7

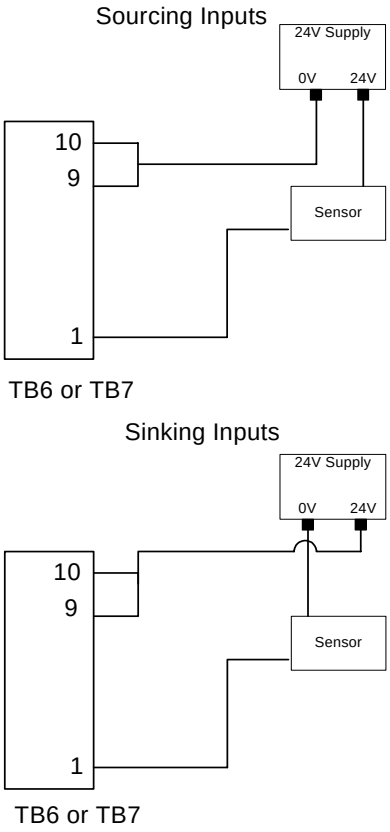


Diagram 8